PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
H04J 3/14, 3/12, H04Q 11/04

A1

(11) International Publication Number: **WO 98/39866**

(43) International Publication Date: 11 September 1998 (11.09.98)

(21) International Application Number: PCT/US98/04271

(22) International Filing Date: 4 March 1998 (04.03.98)

(30) Priority Data:

60/040,487	7 March 1997 (07.03.97)	US
08/816,699	13 March 1997 (13.03.97)	US

(71) Applicant: 3COM CORPORATION [US/US]; 8100 North McCormick Boulevard, Skokie, IL 60076 (US).

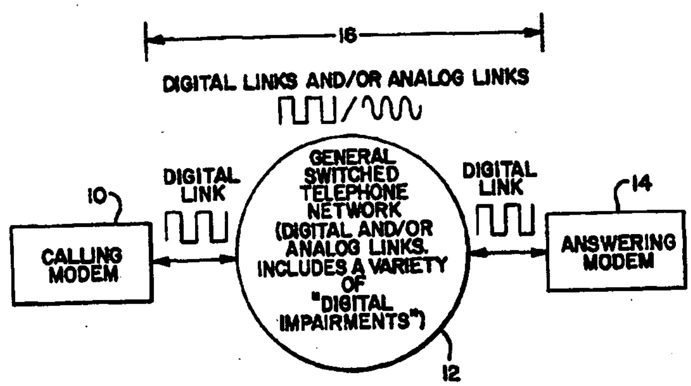
(72) Inventors: DAVENPORT, Bert, A.; 1468 W. Balmoral #4, Chicago, IL 60640 (US). RENKEL, James, A.; 3 S 624 West Avenue, Warrenville, IL 60555-3242 (US). JANKUS, Peter, P.; 5837 W. Lawrence Avenue, Chicago, IL 60630 (US).

(74) Agent: SAMPSON, Matthew, J.; McDonnell Boehnen Hulbert & Berghoff, 300 South Wacker Drive, Chicago, IL 60606 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: **SYSTEM AND METHOD FOR DETERMINING END-TO-END CHARACTERISTICS OF A DATA COMMUNICATION CHANNEL**



(57) Abstract

A method for determining characteristics of a data communication channel between first and second data communication devices. The method includes the step of sending a relatively low power digital probe signal over said channel from said first to said second data communication devices. The method further includes sending a second digital probe signal corresponding to an analog signal having a relatively high-frequency signal with a time-varying dc component from said first to said second data communication devices. The second data communication device receives a signal, said received signal corresponding to said probe signals sent by said first device. In addition, the second data communication device determines whether said received signal varies from a predetermined standard.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

TITLE: System and Method for Determining End-to-End Characteristics of a Data Communication Channel

Copyright Authorization

Background of the Invention

The present invention relates to a method and apparatus for detecting characteristics of a communication channel. In particular, the present invention may be utilized to determine if a communication channel is digital end-to-end. For example, the present method and apparatus may determine whether an Integrated Services Digital Network ("ISDN") to ISDN connection is all digital. Alternatively, the present method and apparatus may determine whether an ISDN to T1/E1, or T1/E1 to T1/E1, connection is all digital. Impairments in the digital connection may also be detected by the apparatus and method of the present invention.

For ISDN to ISDN connections, some known systems send digital data over a channel that is billed as analog. Such systems do not verify that the channel is actually digital before sending the data. Rather, such systems may simply follow the practice of "just trying" to blindly send digital data over what may be an analog link and "seeing" if it works. If the channel is believed to be digital, but is actually analog, the connection will fail.

For ISDN to T1/E1, or T1/E1 to T1/E1, connections, no known methods or apparatii provide high speed data transfer. If an ISDN to T1/E1, or T1/E1 to T1/E1, call is made with

prior art systems, analog communication is typically provided (speeds up to 33.6 full-duplex). In accordance with standard industry practice, communication occurs as if only over an analog link.

Accordingly, it would be desirable to have an improved method and apparatus for determining characteristics of a communication channel.

SUMMARY OF THE INVENTION

In accordance with the present invention, if the communication channel in an ISDN-to-ISDN connection is determined to be end-to-end digital, the data can be sent and billed as an analog call. In addition, if the communication channel is determined to be analog, then a second call attempt can be made requesting (and paying for) a digital call. Moreover, if an ISDN to T1/E1, or T1/E1 to T1/E1, connection is determined to be all digital, digital data transfer rates of up to 62.666 kbps/B channel full-duplex can be achieved utilizing the method and apparatus of the present invention.

It is an object of the present device and method to provide one or more of the following features or advantages:

Enable digital communication at up to 62.666 kbps full-duplex from ISDN to T1/E1, or T1/E1 to T1/E1;

Enable more robust and faster communication at up to 64kbps full-duplex from ISDN to ISDN (BRI--Basic Rate Interface or PRI--Primary Rate Interface) connections while being billed for an analog call;

Determine whether any point-to-point GSTN (General Switched Telephone Network) communication channel is digital end-to-end;

Determine whether any communication channel is digital end-to-end;

Determine whether a G.711 CODEC (Coder-Decoder) is in the network path;

Determine if robbed bit signaling is present and determines the position and number of the robbed bits on an all-digital network;

Determine whether the network requires a minimum "one's density;"

Determine whether one or more digital attenuator pads are present in the connection;

Use a scrambler to prevent the network from compressing the data;

Use zero byte suppression to communicate digitally over a link that converts 00 to 02;

Determine if one or more robbed bits are present in a communication channel; and/or

Determine where one or more robbed bits are located and use 100% of the remaining non-robbed bits for data.

Description of the Drawings

Figure 1 is a block diagram showing a connection between two devices having digital connectivity to the telephone network.

Figure 2 is a block diagram showing a sequence of initial negotiations between the calling device and the answering device shown in Figure 1.

Figure 3 is a block diagram showing more fully the content of block FF in Figure 2.

Figure 4 is an expanded diagram of the contents of block SQA in Figure 2.

Figures 5A and 5B are an expanded diagram of the contents of block SP1 in Figure 2 and a tabular description of the contents, respectively.

Figure 6 is an expanded diagram of block E in Figure 2.

Figure 7 is a block diagram of a transmitter in the calling device and the answering device shown in Figure 1.

Detailed Description of the Preferred Embodiment

Figure 1 is a block diagram showing a connection between two devices having digital connectivity to the telephone network. A calling device 10 has a digital connection to the General Switched Telephone Network ("GSTN") 12. An answering device 14 also has a digital connection to the GSTN 12. At the beginning of a call between the calling device 10 and the answering device 14, certain characteristics of the GSTN are unknown.

Figure 2 shows the negotiation between the calling and answering devices 10 and 14 using the present system. The calling and answering devices are preferably modems. The following is a description of what each of the modems 10, 14 is doing during negotiation over data communication channel 16, as shown in Figure 1.

Figure 1 shows a calling device 10 communicating over a data communication line 16 with an answering device 14. The line 16 may or may not include an analog segment. The devices 10,14 utilize the present system to determine the characteristics of the line 16, including whether the line 16 is entirely digital or it includes an analog segment and whether the line includes any digital impairments.

As used herein, digital impairments refers to limitations on the content of the data sent over the line, such as robbed bit signaling (RBS) or a restricted channel. With an RBS impairment, the an entity, such as a telephone company, operating the line utilizes the least significant bit of selected (e.g., every sixth) byte for overhead purposes. With a restricted channel (rather than a nonrestricted channel, the data sent over the line must have a minimum one's density: there is a limitation on the number of zeros that may validly be placed in a consecutive series. (In some telecommunication systems, such ones density is used to ensure synchronization.)

In addition to digital impairments, regulatory bodies may impose a limitation on the maximum power that may be carried by the line 16. Such power limitations may restrict the energy (in terms of power) that a modem may transmit/receive during various time intervals of predetermined length, such as 250 milliseconds or 3 seconds.

Calling modem procedures

As shown in Figure 2, the calling modem will transmit all marks (bytes of FF hex) until it receives the SQA sequence from the answering modem. The composition of the SQA sequence is shown in Figure 4. The SQA sequence will allow the modem to determine the following:

Is the channel partially analog, or is it all digital?

If the channel is digital, what kinds of "digital impairments" does it contain?

How many and what are the locations of the RBS (robbed bit signaling) bits?

Is the channel restricted or unrestricted (i.e. does it have a minimum one's density requirement?

Does the channel have digital pads in it? If so, what kinds of pads are they?

Referring again to Figure 2, upon receiving the SQA sequence from the answering modem, the calling modem will transmit the SQA sequence until it receives the SP1 sequence from the answering modem.

The SQA sequence, or probing signal, is shown more fully in Figure 4. The sequence may thus be divided into three subparts: (1) 7+6n bytes of 7E, where n is preferably 290; (2) 7 bytes of 00, and (3) a sequence of digital codewords, which corresponds to an analog signal with a 4,000 Hz frequency, ½ the maximum allowable amplitude, and an increasing dc offset voltage. The digital codewords are preferably eight bit PCM codewords or PCM codes.

The first subpart of SQA, 7E in Figure 4, is a low power transmission. This allows the probing signal to meet the maximum power constraints imposed on the line by decreasing the average power transmitted during the SQA sequence. Also, the first subpart effectively "pads" the length of the probing signal, such that the length of the entire probing signal (all three subparts) is a multiple of 6 bytes. In this way, RBS may be more easily detected, since, in many environments, RBS occurs only in every sixth byte at DS0. Moreover, since the SQA sequence is known to the answering device and has a predetermined length, the location and number of robbed bits are determined from the SQA sequence.

The second subpart of SQA, 00 in Figure 4, is sent by the calling modem to test whether the line is a restricted channel or a clear channel. The answering modem effectively knows that if the received signal includes ones in the portion corresponding to the second subpart of the probing signal, the telephone company has probably inserted ones in order to maintain a minimum ones density in the transmitted data. Thus, in such a case, the channel is restricted.

There may be several different one's density requirements imposed by differing transmission technologies in the underlying DS1, DS2, DS3, etc. For example, a very restrictive requirement is that no single PCM codeword can contain all zeros. This effectively limits the number of consecutive zeros to seven in the same PCM codeword, or fourteen zeros across two consecutive PCM codewords.

This type of density restriction may be satisfied, along with all other density restrictions, by inserting a 1 after each string of at most seven consecutive zeros. It should be noted that, since the insertion is done before the data is inverted for transmission, this algorithm may be implemented by inserting a zero after each string of at most seven ones. It should further be noted that the "zero bit insertion/deletion" (ZBID) scheme used with

HDLC/SDLC specifies insertion of a zero after six consecutive ones in the data stream to avoid false flag patterns (0, 6 ones, 0). The flag pattern was chosen to be distinguishable from an abort pattern (0, 7 or more ones) and the idle pattern (continuous ones).

Accordingly, if there is either robbed bit signalling present or a one's density requirement, data transfer occurs at rates below 64 Kbps, such as 56 Kbps. Conversely, if the are no robbed bits and no one's density requirement, i.e. a 64 Kbps clear channel, data transfer may occur at 64 Kbps. In accordance with the preferred embodiment, data transfer rates in excess of 56 Kbps may be achieved, even in the face of robbed bit signalling and a one's density requirement.

The increasing dc offset voltage in the third subpart of SQA, 00, FF, ... , 80 in Figure 4, ensures that, for example, where the line utilizes a 256 level quantizer, all 256 levels are utilized in transmitting the probing signal. The quantizer may be for example a μ-law quantizer, as is used in North America and Japan, or an A-law quantizer , as is used in Europe. In the digital domain, each of the quantizer levels may be represented by a different eight bit codeword. If the signal received by the answering device lacks the full range of the transmitted probing signal, the answering modem may effectively "know" of the digital coding system used in the line.

For example, the presence of a digital attenuator pad in the connection will cause the codewords in the third subpart of SQA to be adjusted in a fixed manner. The answering device, which knows what the third subpart should be, analyzes the received signal and may thereby identify the presence and characteristics of the pad.

The relatively high frequency and amplitude of the probe signal in the third subpart are difficult for an analog converter to translate. Indeed, in many telephone company lines, a filter substantially attenuates signals even approaching 4,000 Hz. If the received signal does

not correspond to the transmitted probe signal, the answering modem may know that an analog segment lies in the line.

The total length of the probing signal used in the preferred embodiment is 256 bytes + 7 bytes + 7 bytes + (290 * 6), or 2010, bytes long. The answering device monitors, during the entire predetermined probing signal, to detect whether the received signal has the least significant bit different than the least significant bit of the probe signal. If so, the answering modem effectively knows that the telephone company line utilizes RBS. In one embodiment, the modems then simply know to communicate at 56 kbps rather than, for example, 64 kbps. In other embodiments, the modems simply do not send data in the least significant bit ("LSB") of any byte because of RBS. The data transmission speed of such modems is accordingly limited to a top speed of 56 kbps.

Figure 5 is an expanded diagram of the contents of block SP1 in Figure 2. The SP1 sequence is an indication of capabilities of the modem and a request to turn on or off various features and speeds in the modem. Upon receiving the SP1 sequence from the answering modem, the calling modem will transmit the SP1 sequence until it receives the E sequence.

Figure 6 is an expanded diagram of block E in Figure 2. Upon receiving the E sequence from the answering modem, the calling modem will transmit the E sequence (fixed length of 5 bytes) followed by data. Upon receiving the data from the answering modem, the calling modem will unclamp its receive data and will proceed to receive data from the answering modem.

Answering modem procedures:

The answering modem will transmit the SQA sequence until it receives the SQA sequence form the calling modem. The SQA sequence will allow the modem to determine the following:

Is the channel partially analog, or is it all digital?

If the channel is digital, what kinds of "digital impairments" does it contain?

How many and what are the locations of the RBS (robbed bit signaling) bits?

Is the channel restricted or unrestricted (i.e. does it have a minimum one's density requirement?

Does the channel have digital pads in it? If so, what kinds of pads are they?

Upon receiving the SQA sequence from the calling modem, the answering modem will transmit the SP1 sequence until it receives the SP1 sequence from the calling modem. The SP1 sequence is an indication of capabilities and of the modem and a request to turn on or off various features and speeds in the modem. Upon receiving the SP1 sequence from the calling modem, the answering modem will transmit the E sequence (fixed length of 5 bytes) followed by data. Upon receiving the data from the calling modem, the calling modem will unclamp its receive data and will proceed to receive data from the calling modem.

One aspect of the present system relates to the detection stage: by sending the SQA pattern, the present system has the ability to:

1) detect if the channel is digital end-to-end

2) detect the exact number and location of RBS

3) detect if digital pads are present (A pad is an adjustment of a digital signal, in one of a predetermined number of different ways, to emulate or mimic the attenuation that would have occurred if the signal had been sent on an analog line rather than a digital line. By noting the systematic padding of the probe signal, the modems may adjust their transmission characteristics to compensate for such padding).

4) detect if the channel is restricted/unrestricted (minimum 1's density issues)

5) detect any digital impairments that we are currently unaware of (since the system sends all 256 codes).

6) detect all of these things without violating transmit power requirements of -15dBm transmit level measured over 3 seconds and 0dBm measured over 250ms.

Another aspect of the invention relates to the data phase: how can the modems may transmit the fastest possible speed given the impairments that are on the line/data communication channel. The present system has the ability to:

1) transmit over any end-to-end digital channel (throughout this specification, except whether context may require otherwise, the term transmit may also be used to mean receive or simply operate.")

2) transmit at speeds of 56K or faster (up to 64K) including intermediate speeds of 62,666bps, 61,333bps, 60,000bps, 58,666bps, 57,333bps. Many other systems can not transmit at these intermediate speeds. This is accomplished by sending 7 data bits/byte (+ 1 non-data bit) in any slot where a RBS bit is present and sending 8 data bits/byte in any slot where a RBS bit is not present.

3) transmit over digital channels that are restricted (i.e. they have a minimum one's density requirement). We do this with zero byte suppression: any time we get 7 0's in a row, insert a 1.

4) transmit over digital channels that have digital pads. We can do this by using a mapping scheme such that the receiver knows that transmitted codes get mapped into a different set of codes after the pad, so the receiver needs to "undo" this mapping. And, any codes that are doubly mapped (i.e. 2 or more transmitted codes get mapped into the same code after the pad) are not sent by the transmitter.

5) Prevent compression by the network: Sometimes the network will monitor the first 6 seconds of the call to see how compressible the data is, and if it is compressible, it will compress it. We prevent this by transmitting scrambled data which will look like wideband white noise.

6) Minimize overhead bits used by the zero byte suppresser. By scrambling the data into white noise, the number of bytes that require action by the zero byte suppresser are a controlled small number of bytes that is independent of the data that the user or protocol is sending.

The present method can determine the exact number and exact location of the RBS which allows faster data transmission than known methods. This is so because if there is just 1 bit robbed by RBS, the preferred method and apparatus described herein may transmit at up to 62,666bps. In fact, with any number or robbed bits between 1 and 5, the preferred method and apparatus provide data transfer rates in excess of 56 Kbps. In a typical network connection, 0-3 robbed bits may be encountered. Finding 4-6 robbed bits appears to be fairly rare.

With respect to the ones density requirement, the present system can detect if we need to avoid sending too many 0's in a row. We can compensate for this in several methods. First, we could just transmit at 56K (and set the LSB=1) which will fix the problem. Or, we could run the "zero byte suppresser" which is an algorithm that inserts a 1 into the data stream if it sees 7 0's in a row and we could run at a faster speed. Third, we could run a scrambler on the data which will make the number of times that we send too many zeros (and would make the zero byte suppresser kick in) very controlled (since the output is basically white noise.

The presently described system will detect the "digital impairment" of digital attenuator pads and could compensate for it, running at a reduced speed, but still running

faster than an analog modem. Because the present system sends all 256 PCM codes, if there are any digital impairments that we are currently unaware of, it is very likely that we could detect and handle them with our current SQA sequence.

A typical digital network may see a transmitted data stream as being compressible. If, however, the transmitted data stream is compressed by the network, the data stream will likely be corrupted and the connection may fail. Use of a scrambler as described herein, on the other hand, makes the transmitted data stream appear as wide-band white noise (i.e. uncompressible). Therefore, the network will not compress the transmitted data stream.

In accordance with the presently preferred embodiment, where an ISDN to ISDN (basic rate or primary rate) connection is determined to be all digital, digital data transfer at rates up to 64kbps/B channel full-duplex can be achieved while the customer is billed for an analog call, which is typically billed at a lower rate than a customer would be billed for a digital call. If an ISDN to T1/E1, or T1/E1 to T1/E1, connection is determined to be all digital, digital data transfer rates of up to 62.666 kbps/B channel full-duplex can be achieved.

The method and apparatus of the present invention may be used in association with 56 Kbps, or higher, transmission speed modems, ISDN modems, and rack modem products. In addition, with the inclusion of a scrambler and a zero-byte suppresser, higher data rates can be achieved. With prior art, 56kbps maximum can be achieved over the channel. With current art, up to 64kbps can be achieved.

Finally, the present device provides added robustness to the communication. The scrambler and zero byte suppresser enable the communication to be successful in situations when the prior art would fail to connect.

The operation of a transmitter and a receiver will now be described with reference to Figure 7. Note that each block shown in Figure 7, except for Byte Conversion, may be turned on or off individually to meet the requirements of the particular network channel:

RBS synchronization:

Each modem's transmitter is synchronized with the remote modem's receiver which is synchronized with the robbed bits in that particular network path (if any are present). The transmitter takes advantage of the fact that it knows if a robbed bit is present in each of the six possible time slots. Each time slot is 8 bits (one byte) long, and the six time slots are periodic i.e., if there is a robed bit in the first time slot, there will be a robbed bit in every sixth byte thereafter (in the same time slot) and these bits together we call "one robbed bit". So, there is a possibility of having from 0-6 robbed bits.

In any time slot where there is a robbed bit, the transmitter will only transmit 7 bits of user data. The 8th bit will be forced to binary one and will be placed in the position of the robbed bit (this happens in the Byte Conversion routine). With this method, the 7 bits of user data will not be corrupted by the RBS, only the 8th bit (not user data) will be corrupted.

In any time slot where there is not a robbed bit, the transmitter will transmit the full 8 bits of user data. These bits will not be corrupted by RBS because we know that it is not present in these time slots. Because we are synchronized with the robbed bit signaling (RBS) in the network and we know the exact number and location of the robbed bits, we can take full advantage of all available information bandwidth in the digital channel.

Scrambler:

The scrambler takes the output of the previous section (which will be either 7 or 8 bits) and scrambles it using standard scrambling techniques to create wide-band white noise energy. In certain networks, the network will monitor the call for a duration of time to

determine if the data is compressible. If it is, the network will compress the data. This would entirely corrupt the communication between two digital modems. To prevent the network from turning on the compressors, we can scramble the data. In addition, the scrambler will create a statistically controlled output which will minimize the output of the Zero Byte Suppresser.

Zero Byte Suppression:

The zero byte suppresser takes the output from the previous section (which will be either 7 or 8 bits) and runs it through the following algorithm: If there are ever 7 bits of "binary zero" in a row, insert a "binary one" into the data stream. This algorithm ensures that minimum one's density requirements are met in the network. Note that this routine outputs the same number of bits that it received as an input and it buffers any additional data caused by the bit insertion. The buffered data is combined with the input of the next byte. If there is ever enough data buffered to transmit entirely out of the buffer, that action will be performed.

Byte Conversion:

The byte conversion routine will output 8 bits of data. If its input was 8 bits, it passes the input data to its output without changing it. If its input was 7 bits, it will insert a "binary one" into the byte at the location where the robbed bit will be, and it will output the 8 bits.

Pad Mapper:

The pad mapper takes the output from the previous section (which will be 8 bits) and it maps the data in such a way as to avoid sending any PCM codes that (due to the digital pad) would result in an ambiguous code at the receiver. Note that if the Zero Byte Suppression and the Pad Mapper are turned on at the same time, they will need to share information to ensure that their respective functions are achieved.

The final output is transmitted onto a digital link on the General Switched Telephone Network. The receiver of the remote modem will have knowledge of which of the above blocks are turned on in the transmitter and knowledge of any additional specific information about the transmitter's configuration that it needs in order to reverse the operations of the transmitter and decode the data.

The description above is sufficient to enable one of ordinary skill in the art to implement the present invention. Nonetheless, to provide additional details regarding the present system, an assembler code listing for a method of implementing certain aspects of the presently described device is provided below. The code has been written for use with a Texas Instruments' TMS320C51 digital signal processor. A User's Guide for the TMS320C5x series of processors is readily available to those of ordinary skill in the art and may be useful to the novice in understanding the commands set forth below.

```
* * * * * * * * * * * * * * * * * * * * * * * * *
*                                               *
*       ISDNLOOP.ASM                            *
*       Contains Main digital loop and other    *
*       routines that are required only during  *
*       a V.120 or V.110 call.                  *
*                                               *
*       Author: Bert A. Davenport               *
*       © 1997, U.S. Robotics Access Corp.      *
* * * * * * * * * * * * * * * * * * * * * * * * *


thresh_7ea    .equ   007    ; # of 7E's in a row to auto-detect speed
tx_7e_15db    .equ   7+290*6        ; # of 7e's to transmit to make -15db power
                                    ; note: this MUST be a 7 + a multiple of 6

*************************************************************************
**
*       orig_isdn & answ_isdn
*
*       Similar to originate_now & answer_now commands from the supervisor
*       excect that these are used to originate or answer digital calls, that
*       is, not PCM data. This is used to establish a 56-64Kps connection over
*       a single channel. Data transported over such a link may be bit
*       transparent, an HDLC-framed protocol (i.e V.120, PPP), V.110 encoding,
*       bonding, etc. A single background loop exists for all data modes as
*       those previously described. The background loop is essentially
*       identical to the sync_idle_t1 loop used in "analog" mode except no
*       interpolation or PCM conversion is necessary. The txvect and rxvect
*       vectors point to the appropriate encoder/decoder for the active
*       digital "mode". The default digital "mode" of operation is 64Kbps
*       clear channel data, although this may be altered as follows;
*
*       Inputs: (exparm)
*               b0      0 = 64 Kbps, 1 = 56Kbps
*               b1      0 = V.120/PPP, 1 = V.110
*               b2      0 = use specified rate, 1 = autodetet rate
*               b3      Do NOT use this bit in exparm
*                       It is set in isdnflg as 1=monitor_line mode
*               b4      0 = no stos, 1 = server to server mode (PCM code)
*               b5      0 = set up call, 1 = tear down call (hangup/reset)
*               b6      0 = single data call (use MB1), 1 = 2 data calls
*               b7      0 = Use IOM Ch 1, 1 = Use IOM Ch 2
*               b8      Reserved; (0 = answer, 1 = originate forced in code)
*               b9      Reserved; (0=no scr, 1 = scrambler for stos)
*               b10     Reserved; (0= no zbs, 1 = zero byte supression stos)
*               b11     Reserved
*               b12     Reserved
*               b13     Reserved
*               b14     Reserved
```

```
*               b15    Reserved
*               Note: for V.110, b8-b15 indicates rate
*
*       Outputs: None
*
***************************************************************************
**

answ_isdn               ;dp = 0
        lacc    #0
        b       comm_isdn
orig_isdn               ;dp = 0
        lacc    #100h           ; or on originate bit (non-v110 only)
comm_isdn
        sacb                    ; accb = answ/originate status
        lacc    exparm                  ; Is this call on Ch 2 & we can have 2 calls?

        and     #o2data+och2 ;
        sub     #o2data+och2 ;

        ldp     #isdn_dp        ; use Ch 1/MB1 data page
        sst     #0,tempx        ; save dp, arp in tempx(dp0) (isdn_dp)
        ldp     #6              ; use Ch 1/MB1 data page
        sst     #0,tempy        ; save dp, arp in tempx(dp0) (dp = 6)
        ldp     #7
        sst     #0,tempz        ; save dp, arp in tempy(dp0) (dp = 7)
        opl     #odigisdn,sysflg2 ; Set digital ISDN bit (V120, ppp,...)

        splk    #dtx_clear_orig_mb1,txvect ; orig digital TX sample vector
        splk    #drx_clear_orig_mb1,rxvect ; orig digital RX sample vector
        bcnd    chk_iflg,neq    ; b if 1 data call or if on Ch 1

        ldp     #isdn_dp2       ; yes, use MB2 data page
        splk    #dtx_clear_orig_mb2,txvect ; orig digital TX sample vector
        splk    #drx_clear_orig_mb2,rxvect ; orig digital RX sample vector
        sst     #0,tempx        ; save dp, arp in tempx(dp0) (isdn_dp2)
        sst     #0,tempy        ; save dp, arp in tempy(dp0) (isdn_dp2)
        sst     #0,tempz        ; save dp, arp in tempy(dp0) (isdn_dp2)

chk_iflg                        ; dp = 7 (Ch 1) or isdn_dp2 (Ch 2)
        ldp     #0
        bit     exparm,f2data ; 2 data calls possible (mlppp)?
        ldp     #7
        xc      2,tc            ; Multilink PPP mode?
         opl    #omlppp,sysflg2 ; yes, or on bit in sysflg2
        lamm    tempx
        sacl    dp_i            ; save isdn_dp or isdn_dp2 depending on chan
        lamm    tempy
        sacl    dp_6            ; save dp 6 or isdn_dp2 depending on chan
        lamm    tempz
```

```
        sacl    dp_7            ; save dp 7 or isdn_dp2 depending on chan
        lst     #0,dp_7                 ; set dp = 7 or isdn_dp2 ar(arp) = ar1
                                ; dp = 7 or isdn_dp2
        lamm    exparm
        orb                     ; turn on orig/answ bit (removed for v.110)
        sacl    isdnflg         ; automode flags (digital)
        sacl    isdnflg_orig    ; automode flags (won't be modified)

        bit     isdnflg,fhangup                 ; Is the hangup bit set?
        bcnd    hangup_call,tc          ; b if yes => hangup call

        .if     PCM code
        bit     isdnflg_orig,fistos ; PCM code server-to-server mode?
        bcnd    no_stos,ntc     ; b if not PCM code stos mode

        splk    #drx_sqa,rxvect         ; SQA receiver
        splk    #rx_7e_init,rxvect64  ; find 1st 0 in 7e patt at 64kbps
        splk    #0,i11_cnt      ; save no data until after 7e's
        splk    #41h,rx_rate    ; init rate mask to 56 & 64K only
        lacc    #24000                  ; 3 second timeout
        samm    dlycnt

        opl     #ostosscr,isdnflg ; force scrambler bit on until supv uses it
        opl     #ostosscr,isdnflg_orig ; scrambler bit on until supv uses it

        lacc    #stos_failed    ; fall back to v.34 if stos fails
        samm    rvec

        bit     isdnflg_orig,forig ; originating?
        splk    #dtx_stos,txvect ; SQA transmitter (answering side)
        xc      2,tc
       splk     #tx_mark,txvect         ; xmit ff (originating side)

        splk    #tx_7e_15db,tx_cnt_sqad         ; send 7 7e's + 6n 7e's
        bd      start_digital_call
         splk   #sqa_7e,tx_sqavect ;init state vector for next state

no_stos
        .endif

        bit     isdnflg,fautod          ; Is the autodetect speed bit set?
        bcnd    init_autod,tc           ; b if yes

        bit     isdnflg,fv110           ; Is the v110 interworking bit set?
        cc      InitV110Machine,tc      ; yes, init V.110

        b       start_digital_call

switch_tx                       ; switch xmit rate
                                ; dp = 7 or isdn_dp2 ar(arp) = ar1
```

```
        xpl     #o56k,isdnflg ; switch rate
        bit     isdnflg,f56k  ; Is the 56K interworking bit set?
        lacc    tx_auto_thrsh ; acc = xmit threshold for next switch
        xc      1,NTC         ; If 64kbps, double # of bytes to send
         add    tx_auto_thrsh ; before next switch
        bcndd   init_speed,unc ; initialize speed related parameters
         sacl   tx_auto_thrsh ; save updated threshold
         sacl   tx_auto_cnt   ; initialize counter

******************************************************************************
**
*       dtx_stos sends 7 7E's followed by the pattern 0,255,1,254,2,253...
*       126,129,127,128; then repeats with 7 7E's and the pattern again
*       This is a speed detection routine for digital server to server mode.
*       This is a possible state of txvect/2
*
*       inputs  dp = 7
*       ar(arp) = ar1
*               ar1 = *(8 bit sample to transmit to the link)
*
*       outputsdp = 7
*               ar(arp) = ar1
******************************************************************************
**
        .if     PCM code
        dtx_stos                        ; dp = 7
        lst     #0,dp_7                 ; set dp = 7 or isdn_dp2 ar(arp) = ar1

        lacc    tx_sqavect
        cala                    ; vector to appropriate routine
                                ; acc = tx data on rtn

*       call    bit_reverse8    ; Prepare "digital" sample for transmission
        retd
         ldp    #7              ; set dp = 7 for return to main loop
         sacl   *               ; txdata_main/aux = txdata

******************************************************************************
**
*       Send 7 7E's + 6n 7e's (n = # needed to make power <-15dBm over 250ms)
******************************************************************************
**
sqa_7e
        lacc    tx_cnt_sqad     ; acc = counter for 7e's
        sub     #1              ; transmit 7e's
        sacl    tx_cnt_sqad     ; decrement and save cntr
        bcnd    no_init_sq0,neq         ; b to not init sq0 sequence

        splk    #7,tx_cnt_sqad          ; init cntr to send 7 00's in a row
        splk    #sqa_00,tx_sqavect ;init state vector for next state
```

```
no_init_sq0
        lacl    #7eh            ; transmit 7e at 64kbps
        ret

***************************************************************************
**
*       Send 7 00's
***************************************************************************
**
sqa_00
        lacc    tx_cnt_sqad     ; acc = counter for 00's
        sub     #1              ; transmit 00's
        sacl    tx_cnt_sqad     ; decrement and save cntr
        bcnd    no_init_sqa,neq         ; b to not init sqa sequence

        splk    #0,tx_cnt_sqau          ; init up cntr to 0 (cnts up to 127)
        splk    #255,tx_cnt_sqad ; init down cntr to 255 (cnts down to 128)
        splk    #sqau_patt,tx_sqavect ;init state vector for next state

no_init_sqa
        lacl    #00h            ; transmit 00 at 64kbps
        ret


***************************************************************************
**
*       Send 00,01,02...127
***************************************************************************
**
sqau_patt                       ; transmit sqa cnt up pattern at 64Kbps
        lacc    tx_cnt_sqau     ; acc = up cntr (cnts up from 0-127)
        sacb                    ; accb = cntr
        add     #1              ; incr cntr
        sacl    tx_cnt_sqau     ; save incr'd up cntr
        lacb                    ; acc = pre-decremented cntr
        retd
        splk    #sqad_patt,tx_sqavect ;init state vector for next state

***************************************************************************
**
*       Send 255,254,253...128
***************************************************************************
**
sqad_patt                       ; transmit sqa cnt down pattern at 64Kbps
        lacc    tx_cnt_sqad     ; acc = down cntr (cnts down from 255-128)
        sacb                    ; accb = cntr
        sub     #1              ; decr cntr
        sacl    tx_cnt_sqad     ; save decr'd cntr
```

```
        sub     #127            ; done with pattern? (return to 7e)
        bcnd    continue_sqa,neq ; b to continue pattern

        lacb                    ; acc = pre-decremented cntr
        splk    #tx_7e_15db,tx_cnt_sqad     ; send 7 7e's + 6n 7e's
        retd
        splk    #sqa_7e,tx_sqavect ;init state vector for next state

continue_sqa
        lacb                    ; acc = pre-decremented cntr
        retd
        splk    #sqau_patt,tx_sqavect ;init state vector for next state


****************************************************************************
**
*       Send SP1 sequence
*       Sequence consists of ff,81,81,d1,e1,f1,g1,h1,i1,j1,k1,l1
*       81's are to sync up to (distinguishing from SQA pattern
*       d1-h1: MSB is 0, lsb is 1, 6 bits of information
*       d1: version information: 000000
*       e1: reserved for future use: 000000
*       f1: rbs position: 1 = rbs on this bit, 0 = no rbs on this bit
*       g1: restricted/unrestricted: 00vwxy
*               v = 1 1 bit insrt 0, no ins (if 1 side selects, both must use)
*               w = 1 scrambler 0, no scram (if 1 side selects, both must use)
*               x = 1 restricted, 0, unrestr
*               y = 1, 64Kbps speed selected, 0 = not 64Kbps speed selected
*       h1: speed bits: mnopqr
*          m: 1 = 62.6Kbps selected, 0 = not selected
*          n: 1 = 61.3Kbps selected, 0 = not selected
*          o: 1 = 60.0Kbps selected, 0 = not selected
*          p: 1 = 58.6Kbps selected, 0 = not selected
*          q: 1 = 57.3Kbps selected, 0 = not selected
*          r: 1 = 56.0Kbps selected, 0 = not selected
*       i1: CRC 4 bits: 000CRCa1 (4 bits)
*       j1: CRC 4 bits: 000CRCb1 (4 bits)
*       k1: CRC 4 bits: 000CRCd1 (4 bits)
*       l1: CRC 4 bits: 000CRCc1 (4 bits)
*
****************************************************************************
**
tx_sp1a
        splk    #0ffffh,tx_crc  ; init SDLC CRC (for next time we go in-frame)
        lacc    #0ffh           ; TX first byte of sp1 (sync byte)
        retd
        splk    #tx_sp1b,tx_sqavect ;init state vector for next state

tx_sp1b
        lacc    #81h            ; TX first byte of sp1 (sync byte)
```

```
        retd
         splk   #tx_sp1c,tx_sqavect ;init state vector for next state

tx_sp1c
        lacc    #81h            ; TX 2nd byte of sp1 (sync byte)
        retd
         splk   #tx_sp1d,tx_sqavect ;init state vector for next state

tx_sp1d
        lacc    #01h            ; TX 3rd byte of sp1 (version byte)
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#txinfo0_buf
        bd      do_tx_crc
         splk   #tx_sp1e,tx_sqavect ;init state vector for next state

tx_sp1e
        lacc    #01h            ; TX 4th byte of sp1 (reserved byte)
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#txinfo0_buf+1
        bd      do_tx_crc
         splk   #tx_sp1f,tx_sqavect ;init state vector for next state

tx_sp1f
        lacc    rbs_map,1       ; TX 5th byte of sp1 (RBS info byte)
        and     #7eh            ; force to 6 bits
        or      #1              ; force on lsb
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#txinfo0_buf+2
        bd      do_tx_crc
         splk   #tx_sp1g,tx_sqavect ;init state vector for next state

tx_sp1g
        lacc    rbs_map                 ; TX 6th byte of sp1 (restricted info byte)
        splk    #0,tempx        ; assume no 64K
        xc      2,eq            ; can we do 64K (no rbs and unrestricted)?
         splk   #2,tempx        ; turn on 64K bit

                                ; turn on scrambler any time we do 64K
                                ; if it's enabled

        bit     isdnflg,fstosscr ; is local copy of scrambler on already?
        bcnd    no_scrg,ntc     ; b to not use scrambler if not enabled

        bcnd    no_scrg,neq     ; b to not use scrambler if not 64K

        opl     #8,tempx        ; turn on scrambler bit in tx_sp1

no_scrg
        and     #40h            ; mask off all but restricted info bit
        and     rx_rate         ; mask with rate mask
```

```
        bsar    4               ; position bit in 04h position
        or      tempx           ; or on 64K bit
        or      #1              ; force on lsb
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#txinfo0_buf+3
        bd      do_tx_crc
         splk   #tx_sp1h,tx_sqavect ;init state vector for next state

tx_sp1h
        splk    #4,tx_cnt_sqad          ; init cntr to send 4 nibbles of CRC
        lacc    rx_rate,1       ; TX 7th byte of sp1 (speed byte)
        and     #7eh            ; force rate mask to 6 bits
        or      #1              ; force on lsb
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#txinfo0_buf+4
        bd      do_tx_crc
         splk   #tx_sp1i,tx_sqavect ;init state vector for next state


tx_sp1i                         ; xmit 4 nibbles of CRC
        lacc    tx_crc,1
        cmpl                    ; invert before sending
        and     #01eh           ; mask to 4 bits
        or      #1
        sacb
        lacl    tx_crc          ; shift crc right 1 byte
        bsar    4
        or      #0f000h                 ; crc = fxxx (xx = old high byte of crc)
        sacl    tx_crc

        lacc    tx_cnt_sqad     ; acc = counter for 7e's
        sub     #1              ; transmit 7e's
        sacl    tx_cnt_sqad     ; decrement and save cntr
        bcnd    not_over,neq    ; b to not init sq0 sequence

        splk    #tx_sp1a,tx_sqavect ;init state vector for next state

not_over
        lacb                    ; crc to xmit
        ret



do_tx_crc
        sacb
        calc_crc_byte tx_crc    ; calculate crc and store in tx_crc
        lacb
        ret
```

```
*****************************************************************************
**
*       Send E (ff,81,81,81,81)
*****************************************************************************
**
tx_e
        lacl    #0ffh           ; transmit ff at 64kbps
        splk    #4,tx_cnt_sqad          ; init cntr to send 4 81's in a row
        retd
        splk    #tx_e81,tx_sqavect ;init state vector for next state

tx_e81
        lacc    tx_cnt_sqad     ; acc = counter for 81's
        sub     #1              ; transmit 81's
        sacl    tx_cnt_sqad     ; decrement and save cntr
        bcnd    no_init_scr,neq         ; b to not init scrambler

                                ; done sending E sequence, transmit scr data

        bit     isdnflg,forig   ; originating?
        bcnd    tx_e_orig,tc    ; b if originating
                                ; (answering side) tx scr data, rx E

        bit     rx_rate,fs64    ; is 64kbps possible?
        cc      connect_64,tc   ; b if yes

        bit     rx_rate,fs64    ; is 64kbps possible?
        bcnd    no_chk56t,tc    ; yes, don't check other speeds

        bit     rx_rate,fs56    ; is 56kbps possible?
        cc      connect_56,tc   ; b if yes

no_chk56t

        lacc    rxvect
        sacl    rxvect_bak      ; back up receiver to restore later

        splk    #drx_sqa,rxvect         ; SQA receiver
        splk    #rx_e,rxvect64 ; receive E sequence (answering side)

        lacl    #81h            ; transmit 81 at 64kbps
        ret

tx_e_orig                       ; originator: already receiving scr data
                                ; now tx scr data

        lacc    txvect_bak
        sacl    txvect          ; restore txvect to tx scr data

no_init_scr
```

```
        lacl    #81h            ; transmit 81 at 64kbps
        ret


****************************************************************************
**
*       drx_sqa receives 7 7E's followed by the pattern 0,255,1,254,2,253...
*       126,129,127,128; then repeats with 7 7E's and the pattern again
*       This is a speed detection routine for PCM code server to server mode.
*       This is a possible state of rxvect/2
*
*       inputs  dp = 7
*       ar(arp) = ar1
*               ar1 = *(8 bit rceive sample from the link)
*
*       outputsdp = ?
*               ar(arp) = ar1
****************************************************************************
**

drx_sqa
                                ; dp = 7, ar(arp) = ar1
                                ; don't disturb AR1, it has rcv data in it

*       lacc    *               ; acc = 8 bit recceive data
*       call    bit_reverse8    ; Modify rcv samp for main (digital) channel
*       sacl    *               ; save reversed digital data

        call    handshake       ; time out and fall back to V.34?

        lacc    #b0             ; start processing at b0 and go through b7
        samm    treg2           ; save into dynamic bit pointer test register
        lacc    #b8             ; 64K ends before b8 (b0 through b7)
        sacl    tempw           ; tempw = bit to stop at

        lacl    rxvect64        ; process rcv samp looking for 64kbps pattern
        cala

        ret

****************************************************************************
**
*       Increment rbs position modulo 6 (0-5)
****************************************************************************
**
incr_rbs
        lacc    rx_cnt_rbs      ; increment rbs position (0-5)
        add     #1              ; rbs repeats every 6 bytes
        sacl    rx_cnt_rbs
        sub     #6              ; force to be modulo 6 (if 6, set to 0)
```

```
        retc    neq
        sacl    rx_cnt_rbs
        ret

****************************************************************************
**
*       RBS bit found, turn bit on in rbs_map
****************************************************************************
**
set_rbs                         ; bit is 1 (7F)
        lacc    #1
        rpt     rx_cnt_rbs      ; position 1 bit in mask
        sfl                     ; if rx_cnt_rbs=0, lsb rbs_map=1
        bsar    1
        retd
        or      rbs_map                 ; if rx_cnt_rbs=5, rbs_map=10h
        sacl    rbs_map                 ; turn RBS bit position on in map

****************************************************************************
**
*       Check if need to save data for I11 screen (used by rx_sqa)
****************************************************************************
**
chk_i11
        lacc    i11_cnt                 ; decrement i11 counter
        sub     #1
        retc    lt              ; do nothing if cntr = 0

        sacl    i11_cnt

        lacc    *,ar2           ; save info for I11
        and     #0ffh
        orb
        b       save_i11

****************************************************************************
**
*       Look for 0,255,1,254...127,128 pattern (up cntr state 0,1,2...)
****************************************************************************
**
rx_sqau
        call    incr_rbs        ; increment rbs position

        lacc    #0200h
        sacb
        call    chk_i11                 ; check on saving data to i11

        lacc    *               ; acc = rx samp
        sub     rx_cnt_sqau     ; is rx samp = cntr?
        bcnd    same_8bits,eq ; b if equal
```

```
        lacc    *               ; acc = rx samp (eliminate lsb)
        or      #1
        bldd    #rx_cnt_sqau,tempx ; rxvect = rxvect1 (initialize rxvect)
        opl     #1,tempx

        sub     tempx           ; is rx samp = cntr? (eliminate lsb)
        bcnd    same_7bits,eq ; b if equal

        lacc    rx_cnt_sqau
        bcnd    bad_pattern,neq         ; bad pattern if expected samp <> 0

        lacc    *               ; acc = actual rx samp
        sub     #2
        bcnd    bad_pattern,neq         ; bad pattern if expected samp <> 2

        bd      same_8bits      ; pattern OK
        opl     #40h,rbs_map ; or msb of lower byte on => 0 to 2 conversion

bad_pattern
        retd
        splk    #rx_7e_init,rxvect64 ; reset pattern detection from scratch

same_7bits
        call    set_rbs         ; rbs detected, turn bit on in rbs_map

same_8bits
        lacc    rx_cnt_sqau     ; increment counter
        add     #1
        sacl    rx_cnt_sqau
        retd
        splk    #rx_sqad,rxvect64 ; next state of receiver (cnt down)

****************************************************************************
**
*       Look for 0,255,1,254...127,128 pattern (down cntr state 255,254,...)
****************************************************************************
**
rx_sqad
        call    incr_rbs        ; increment rbs position

        lacc    #0200h
        sacb
        call    chk_i11                 ; check on saving data to i11

        lacc    *               ; acc = rx samp
        sub     rx_cnt_sqad     ; is rx samp = cntr?
        bcnd    same_8bitsd,eq          ; b if equal
```

```
        lacc    *               ; acc = rx samp (eliminate lsb)
        or      #1
        bldd    #rx_cnt_sqad,tempx ; rxvect = rxvect1 (initialize rxvect)
        opl     #1,tempx

        sub     tempx           ; is rx samp = cntr? (eliminate lsb)
        bcnd    same_7bitsd,eq          ; b if equal

                                ; bad pattern
        retd
         splk   #rx_7e_init,rxvect64 ; reset pattern detection from scratch

same_7bitsd
        call    set_rbs         ; rbs detected, turn bit on in rbs_map

same_8bitsd
        splk    #rx_sqau,rxvect64 ; next state of receiver (cnt up)
        lacc    rx_cnt_sqad     ; increment counter

        sub     #1
        sacl    rx_cnt_sqad
        sub     #127
        retc    neq

                                ; SQA pattern fully detected

        bit     isdnflg,forig   ; originating?
        splk    #tx_sp1a,tx_sqavect ; ; SP1 transmitter (answering side)
        splk    #rx_sp1,rxvect64 ; receive speed 1 sequence (ans/orig)
        retc    ntc

        splk    #tx_7e_15db,tx_cnt_sqad      ; send 7 7e's + 6n 7e's
        splk    #sqa_7e,tx_sqavect ;init state vector for next state
        retd
         splk   #dtx_stos,txvect ; SQA transmitter (originating side)


********************************************************************************
**
*       Receive SP1 sequence
*       Sequence consists of ff,81,81,d1,e1,f1,g1,h1
*       ff,81's are to sync up to (distinguishing from SQA pattern
*       d1-h1: MSB is 0, lsb is 1, 6 bits of information
*       d1: version information: 000000
*       e1: reserved for future use: 000000
*       f1: rbs position: 1 = rbs on this bit, 0 = no rbs on this bit
*       g1: restricted/unrestricted: 00vwxy
*               v = 1 1 bit insrt 0, no ins (if 1 side selects, both must use)
*               w = 1 scrambler 0, no scram (if 1 side selects, both must use)
*               x = 1 restricted, 0, unrestr
```

```
*              y = 1, 64Kbps speed selected, 0 = not 64Kbps speed selected
*         h1: speed bits: mnopqr
*           m: 1 = 62.6Kbps selected, 0 = not selected
*           n: 1 = 61.3Kbps selected, 0 = not selected
*           o: 1 = 60.0Kbps selected, 0 = not selected
*           p: 1 = 58.6Kbps selected, 0 = not selected
*           q: 1 = 57.3Kbps selected, 0 = not selected
*           r: 1 = 56.0Kbps selected, 0 = not selected
*
*****************************************************************************
**



*****************************************************************************
**
*       Receive 81,81
*****************************************************************************
**

rx_sp1
        call    rx_81_i
rx_81
        lacc    *               ; acc = rx samp
        or      #1              ; or on 1 bit (in case lsb destroyed by RBS)
        sub     #81h            ; is rx samp = 81?
        bcnd    yes_81,eq       ; b if equal

                                ; reset pattern detector
rx_81_i
        splk    #0ffffh,rx_crc  ; initialize crc (digital ISDN only)
        splk    #rx_81,rxvect64 ; look for 81 patt
        retd
        splk    #2,cnt_7e_64    ; init # 81's in a row to validate speed

yes_81
        lacc    cnt_7e_64       ; acc = cnt_7e_64 = 81 counter
        sub     #1              ; decrement 81 counter
        sacl    cnt_7e_64       ; save cnt_7e_64
        retc    gt              ; return if counter > 0

        retd
        splk    #rx_ver,rxvect64 ; look for version byte

*****************************************************************************
**
*       Receive version byte
*****************************************************************************
**
rx_ver
```

```
        lacc    *               ; acc = rx samp = rbs byte
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#rxinfo0_buf
        call    do_rx_crc       ; calculate rx crc

        retd
        splk    #rx_xx,rxvect64 ; receive reserved byte

*****************************************************************************
**
*       Receive reserved byte
*****************************************************************************
**
rx_xx
        lacc    *               ; acc = rx samp = rbs byte
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#rxinfo0_buf+1
        call    do_rx_crc       ; calculate rx crc

        retd
        splk    #rx_rbs,rxvect64 ; receive rbs byte


*****************************************************************************
**
*       Receive RBS byte
*****************************************************************************
**
rx_rbs
        lacc    *               ; acc = rx samp = rbs byte
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#rxinfo0_buf+2
        call    do_rx_crc       ; calculate rx crc
        and     #7eh            ; force to 6 bits
        bsar    1               ; shift right 1
        sacl    rbs_map_rx

        retd
        splk    #rx_rst,rxvect64 ; receive restricted data info byte

*****************************************************************************
**
*       Receive restricted info byte
*****************************************************************************
**
rx_rst
        lacc    *               ; acc = rx samp = rbs byte
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#rxinfo0_buf+3
        call    do_rx_crc       ; calculate rx crc
```

```
                        ; turn on scrambler if remote side requests it
        bit     *,frscr         ; is scrambler bit on in restricted byte?
        lacc    *,4             ; acc = rx samp = restricted info byte
        xc      2,ntc           ;
        apl     #astosscr,isdnflg ; force scrambler bit off if remote is off

        and     #40h            ; mask off all but restricted bit
        or      rbs_map_rx
        sacl    rbs_map_rx

        lacc    *,5             ; acc = rx samp
        and     #40h            ; mask off all but 64K speed bit
        or      #0ffbfh
        sacl    rx_rate_rx      ; received rate from remote side

        retd
        splk    #rx_spd,rxvect64 ; receive restricted data info byte

******************************************************************************
**
*       Receive speed info byte
******************************************************************************
**
rx_spd
        lacc    *               ; acc = rx samp = speed byte
        sacl    tempx           ; save for y12 screen
        bldd    tempx,#rxinfo0_buf+4
        call    do_rx_crc       ; calculate rx crc
        and     #7eh            ; mask off all but speed bits
        bsar    1               ; shift right to align with rx_rate
        or      #0ffc0h
        and     rx_rate_rx      ;
        sacl    rx_rate_rx      ; build received rate from remote side

        splk    #4,cnt_7e_64    ; receive 4 bytes w/CRC in it
        splk    #0,rx_crcr      ; init received crc
        retd
        splk    #rx_crcb,rxvect64 ; receive restricted data info byte


******************************************************************************
**
*       Receive 4 crc bytes (000xxxx1) xxxx = 4 bits of crc in each byte
******************************************************************************
**
rx_crcb
        lacc    rx_crcr,16      ; combine with previous CRC nibbles
        sacb
```

```
        zap
        ror                     ; force carry bit to 0
        lacc    *
        and     #1eh            ; mask to 4 CRC bits
        ror                     ; crc = 4 lsb's
        ror                     ; crc = 3 lsb's and carry bit
        orb
        rpt     #3              ; rotate right 4 times
         ror
        sach    rx_crcr         ; save updated received CRC

        lacc    cnt_7e_64       ; acc = cnt_7e_64 = counter
        sub     #1              ; decrement counter
        sacl    cnt_7e_64       ; save cnt_7e_64
        retc    gt              ; return if counter > 0
                                ; received entire crc
        lacl    rx_crcr
        and     #0ffh
        call    do_rx_crc       ; calculate rx crc (low byte)

        lacl    rx_crcr
        bsar    8
        and     #0ffh
        call    do_rx_crc       ; calculate rx crc (high byte)

        cpl     #0f0b8h,rx_crc          ; is CRC OK? (should be f0b8)
        splk    #rx_sp1,rxvect64 ; addr for BAD CRC (start SP1 over)
        retc    ntc             ; BAD CRC, start over


        lacl    rbs_map
        or      rbs_map_rx
        sacl    rbs_map                 ; or local and remote rbs_map's together

        lacl    rx_rate
        and     rx_rate_rx      ; and remote and local rate masks together
        sacl    rx_rate

        bit     isdnflg,forig   ; originating?
        splk    #tx_e,tx_sqavect ; E transmitter (answering side)
        splk    #rx_e,rxvect64 ; receive E sequence (ans/orig)
        retc    ntc

        retd
         splk   #tx_sp1a,tx_sqavect ; ; SP1 transmitter (originating side)


do_rx_crc
        sacb
        calc_crc_byte rx_crc  ; calculate crc and store in rx_crc
```

```
        lacb

        ret

****************************************************************************
**
*       Receive E (81,81,81,81)
****************************************************************************
**

rx_e
        call    rx_81_ie
rx_81e
        lacc    *               ; acc = rx samp
        or      #1              ; or on 1 bit (in case lsb destroyed by RBS)
        sub     #81h            ; is rx samp = 81?
        bcnd    yes_81e,eq      ; b if equal

                                ; reset pattern detector
rx_81_ie
        splk    #rx_81e,rxvect64 ; look for 81 patt
        retd
        splk    #4,cnt_7e_64    ; init # 81's in a row to validate speed

yes_81e
        lacc    cnt_7e_64       ; acc = cnt_7e_64 = 81 counter
        sub     #1              ; decrement 81 counter
        sacl    cnt_7e_64       ; save cnt_7e_64
        retc    gt              ; return if counter > 0

                                ; E sequence received
                                ; connect and prepare for scrambled data


        bit     isdnflg,forig   ; originating?
        bcnd    rx_e_orig,tc    ; b if originating

        lacc    rxvect_bak      ; (answering side) tx already sending scr data
        sacl    rxvect          ; restore receiver to receive scr data
        ret

rx_e_orig                       ; originator: transmit E, receive scr data

        bit     rx_rate,fs64    ; is 64kbps possible?
        cc      connect_64,tc   ; b if yes

        bit     rx_rate,fs64    ; is 64kbps possible?
        bcnd    no_chk56,tc     ; yes, don't check other speeds

        bit     rx_rate,fs56    ; is 56kbps possible?
```

```
        cc      connect_56,tc  ; b if yes

no_chk56
        lacc    txvect
        sacl    txvect_bak      ; save txvect to restore later

        splk    #dtx_stos,txvect ; SQA transmitter (originating side)
        retd                    ; transmit E, receive scrambled data
        splk    #tx_e,tx_sqavect ; E transmitter (originating side)


*****************************************************************************
**
*       Look for 7 7e's (or 7f's)
*****************************************************************************
**
rx_7e_init
        call    rx_7e_i
rx_7e
        call    incr_rbs        ; increment rbs position

rx_7e_00
        lacc    #0400h
        sacb
        call    chk_i11                 ; check on saving data to i11

        lacc    *               ; acc = rx samp
        sub     #7eh            ; is rx samp = 7e?
        bcnd    yes_7e,eq       ; b if equal

        lacc    *               ; acc = rx samp (eliminate lsb)
        sub     #7fh            ; is rx samp = 7f?
        bcnd    yes_7f,eq       ; b if equal
                                ; reset pattern detector
rx_7e_i
        splk    #rx_7e,rxvect64 ; look for 7e/7f patt
        splk    #thresh_7ea,cnt_7e_64 ; init # 7e's in a row to validate speed
        splk    #0,rbs_map      ; init rbs_map to 0 (no RBS)
        splk    #4,rx_cnt_rbs   ; init rbs position cntr to 5

        ret

yes_7f
        call    set_rbs         ; rbs detected, turn bit on in rbs_map

yes_7e

        lacc    cnt_7e_64       ; acc = cnt_7e_64 = cnt_7e (7e counter)
        sub     #1              ; decrement 7e counter
        sacl    cnt_7e_64       ; save cnt_7e_64
```

```
        retc    gt                  ; return if counter > 0

        splk    #rx_00,rxvect64 ; look for 00 patt
        splk    #7,cnt_7e_64 ; find 7 0's in a row

        ret

*******************************************************************************
**
*       Look for 7 00's (or 01's if RBS) (or 02's if restricted data type)
*******************************************************************************
**
rx_00
        call    incr_rbs        ; increment rbs position

        lacc    *               ; acc = rx samp
        bcnd    yes_00,eq       ; b if equal

        sub     #1h             ; is rx samp = 1?
        bcnd    yes_01,eq       ; b if equal

        opl     #40h,rbs_map ; or msb of lower byte on => 0 to 2 conversion

        sub     #1h             ; is rx samp = 2?
        bcnd    yes_00,eq       ; b if equal

        sub     #1h             ; is rx samp = 3?
        bcnd    yes_01,eq       ; b if equal

        splk    #1,cnt_7e_64  ; init # 7e's in a row to validate speed
*       bd      rx_7e_00        ; see if still receiving 7e's
        apl     #0ffbfh,rbs_map         ; and msb of lower byte off => no 0 to 2 conv

        lacc    *               ; acc = rx samp
        sub     #7eh            ; is rx samp = 7e?
        bcnd    yes_7e,eq       ; b if equal

        lacc    *               ; acc = rx samp (eliminate lsb)
        sub     #7fh            ; is rx samp = 7f?
        bcnd    yes_7f,eq       ; b if equal

        call    reset_i11       ; received 7 7e's and then received something
                                ; that is <> 7e/7f and <> 00/01.  Save it.
        lacc    *,ar2           ; save info for I11
        and     #0ffh
        or      #0300h
        call    save_i11

        splk    #18,i11_cnt   ; save 18 bytes of data after <>7e/00 byte
        b       rx_7e_i                 ; reset to look for 7e/7f
```

```
yes_01
        call    set_rbs         ; rbs detected, turn bit on in rbs_map

yes_00

        lacc    cnt_7e_64       ; acc = cnt_7e_64 = cnt_7e (00 counter)
        sub     #7              ; first 00 detected?
        cc      reset_i11,eq    ; b if yes

        lacc    *,ar2           ; save info for I11
        and     #0ffh
        or      #0100h
        call    save_i11

        lacc    cnt_7e_64       ; acc = cnt_7e_64 = cnt_7e (00 counter)
        sub     #1              ; decrement 00 counter
        sacl    cnt_7e_64       ; save cnt_7e_64
        retc    gt              ; return if counter > 0

        splk    #12,i11_cnt     ; save 12 bytes of data after 00's
        splk    #rx_sqau,rxvect64 ; next state of receiver (cnt up)
        splk    #0,rx_cnt_sqau          ; init up cntr to 0 (cnts up to 127)
        splk    #255,rx_cnt_sqad ; init down cntr to 255 (cnts down to 128)

        ret

save_i11
        lar     ar2,#stos_i11
        lar     ar2,*
        sacl    *+,ar1          ; save receive data into vfc_probe_results
        sar     ar2,tempx
        bldd    tempx,#stos_i11
        ret


reset_i11                       ; initialize stos_i11 ptr
        mar     *,ar2
        lar     ar2,#stos_i11
        splk    #vfc_probe_results,*
        lar     ar2,#vfc_probe_results

        rptz    #24
        sacl    *+

        mar     *,ar1

        lacl    #stos_stats     ; report rdata_on to SV
```

```
        b       queue_status


*****************************************************************************
**
*       3 second timeout for stos mode elapsed => fall back to v.34
*****************************************************************************
**
stos_failed
        call    set_amfe

        lar     ar1,#arcflg
        apl     #aarc,*         ; turn off PCM code

        ldp     #6
      splk  #queue_unpacked,rxd_vect  ; use unpacked data mode as default

        ldp     #7
        apl     #adigisdn,sysflg2 ; Turn off digital ISDN bit (V120, ppp,...)

        b       init_retrain_v34


        .endif

connect_isdn                    ; send isdn connect messages to supv
                                ; and unclamp data
                                ; input: sysflg has accurate speed bit
                                ; dp = 7 or isdn_dp2
        mar     *,ar1
        sar     ar1,tempy       ; save ar1 for tx_e (PCM code)

        lar     ar1,#sysflg2
        bit     *,f_usr_mode

        bcnd    not_stos,ntc
         lacc   #connect_arc|8000h    ; report connect

        call    queue_status  ;

        bit     isdnflg,f56k  ; 1 = 56K, 0 = 64K
        lacc    #1010h              ; 64K connect
        xc      2,tc            ; Connect stos 56K?
         lacc   #0e0eh          ; 56K connect

        call    queue_status  ; report speed to supervisor
```

```
        b       connect_com

not_stos

        lacc    #connect_digital|8000h ; report digital connection to supv
        call    queue_status   ;

        lacc    isdnflg        ; acc = isdnflag (contains speed bit
        and     #o56k          ; mask off all bits except speed bit
        xor     #o56k          ; toggle bit
        call    queue_status   ; report speed to supervisor

connect_com

        lacl    #rx_data_on    ; report rdata_on to SV
        call    queue_status

        lacl    #tx_data_on    ; report tdata_on to SV
        call    queue_status

        lacl    #carrier_present
        call    queue_status   ; report carrier present to supervisor

        lar     ar1,#sysflg    ; Point AR1 to the flag
        opl     #otxd|orxd,*   ; and unclamp both Rx and Tx data...

        lar     ar1,tempy      ; restore ar1 for tx_e (PCM code)
        bldd    txvect,#txvect ; txvect(dp7) = txvect(dp7 or isdn_dp2)
        bldd    rxvect,#rxvect ; rxvect(dp7) = rxvect(dp7 or isdn_dp2)
        ldp     #7             ; force dp = 7 on return to main loop
        ret                    ; dp = 7, ar(arp) = ar1

init_speed                     ; initialize data parms for particular speed
                               ; dp = 7 (Ch 1/2) or isdn_dp2 (Ch 2)

        lacl    #5             ; prepare for var init
        sacl    txmark         ; init SDLC marks counter
        sach    txbindx                ; clear data buffer
        sach    txdbuf         ; clear data buffer
        .if     PCM code
        sach    scraml         ; init scrambler to 0
        sach    scramh         ; init scrambler to 0
        .endif
        splk    #get_data,txd_vect ; initialize txd_vect
        splk    #0ffh,txmask   ; mask 8 bits
        bit     isdnflg,f56k   ; Is the 56K interworking bit set?
        lacl    #8             ; Assume 64Kbps interworking (8bit/baud)
        xc      1,TC           ; If yes, then set the number of TX bits
        sub     #1             ; per "baud" to 7 (56Kbps)
```

```
        sacl    txnbit          ; Set appropriate bits/symbol
        xc      2,TC            ; If 56K, then set the number of bits
         splk   #07fh,txmask    ; mask 7 bits (56kbps)

        ldp     #7
        lst     #0,dp_6                 ; set dp = 6 or isdn_dp2 ar(arp) = ar1
        sacl    rxnbit          ; Set appropriate bits/symbol
        sach    rxbindx                 ; clear data buffer
        sach    rxdbuf          ; clear data buffer
        .if     PCM code
        sach    descrh          ; init descrambler to 0
        sach    descrl          ; init descrambler to 0
        .endif
        sach    rx_in_frame     ; init to out of frame, no crc, no filter flgs
        splk    #0ffh,rxmask    ; mask 8 bits (64kbps)
        xc      2,TC            ; If 56K, then set the number of bits
         splk   #07fh,rxmask    ; mask 7 bits (56kbps)

        splk    #filter_marks,rxd_vect  ; use filter_marks data mode as default

        ldp     #7
        lst     #0,dp_7                 ; set dp = 7 or isdn_dp2 ar(arp) = ar1
        ret

init_autod
                                        ; dp = 7 (Ch 1/2) or isdn_dp2 (Ch 2)
        splk    #dtx_auto,txvect        ; send auto-detect pattern for TX
        splk    #drx_auto,rxvect        ; send auto-detect pattern for RX
        splk    #start_over,rxvect64    ; find 1st 0 in 7e patt at 64kbps
        splk    #start_over,rxvect56    ; find 1st 0 in 7e patt at 56kbps

        splk    #0ffffh,cnt_1s_56       ; 0ffffh indicates no 0's received
        lacc    #2000                   ; wait 2000 bytes before switching
        sacl    tx_auto_cnt             ; xmit rate
        sacl    tx_auto_thrsh           ; set reset value

        splk    #32000,cnt_bytes_lo     ; init 32-bit timeout cntr to 4 sec.
        splk    #0,cnt_bytes_hi                 ; init 32-bit timeout cntr to 4 sec.

*****************************************************************************
**
* Digital call initialization (common to both originate and answer modes and
*                       common to all digital data call setup paths)
*****************************************************************************
**

start_digital_call
                                ; dp = 7 (Ch 1/2) or isdn_dp2 (Ch 2)
        call    init_speed      ; initialize xmit & rcv states & vars
```

```
                        ; dp = 7 (Ch 1/2) or isdn_dp2 (Ch 2)

        bit     isdnflg_orig,fistos ; PCM code server-to-server mode?
        lacc    isdnflg         ; acc = isdn flags.  is it v120 fixed rate?
        and     #o110aut        ; is it v120 fixed rate? (0 = yes)
        cc      connect_isdn,eq,ntc ; Yes, send connect messages (rtn dp = 7)
                                ; dp = ?
        ldp     #7
        lst     #0,dp_7                 ; set dp = 7 or isdn_dp2 ar(arp) = ar1

*       Initialize tx and rx front end pointers for ISDN

        pop                             ; skip analog stuff in cmdproc
        bit     isdnflg_orig,fch2       ; Is the command for IOM Ch 2?
        ldp     #isdn_dp                ; set up channel based parameters
        bcnd    set_iom_chan,unc        ; set correct channel for tx/rxvect1/2

****************************************************************************
**
*       dtx_clear_orig_mb1 and dtx_clear_answ_mb1 handle xmit HDLC functions
*       for V.120 and PPP on mailbox 1.
*       This is a possible state of txvect/2
*
*       inputs  dp = 7
*       ar(arp) = ar1
*               ar1 = *(8 bit sample to transmit to the link)
*
*       outputsdp = 7
*               ar(arp) = ar1
****************************************************************************
**

dtx_clear_orig_mb1
dtx_clear_answ_mb1                      ; dp = 7

        sar     ar1,tempx       ; tempx = ptr to xmit sample

        call    get_txd,*,ar1   ; get data from supv. (must not use tempx)
                        .       ; dp = 7

        .if     PCM code
        lacl    txdata
        and     txmask          ; mask off extra bits (for scrambler)
      sacl    txdata          ; save transmit data

        bit     isdnflg,fstosscr ; PCM code server to server call with scrambler?
        cc      scram_v32orig,tc ; call PCM code server to server TX routine

*       bit     isdnflg,fstoszbs ; PCM code stos call w/zero byte supression
*       cc      tx_zbs,tc       ; call tx zero byte supression routine
```

```
        lacl    txdata
        lar     ar1,#arcflg
        bit     *,farc          ; stos?
        cc      bit_reverse8,tc ; Prepare "digital" sample for transmission

        sacl    txdata          ; txdata_main/aux = reversed bits.

        .endif


        lar     ar1,tempx       ; ar1 = ptr to xmit sample

        retd
         lacl   txdata          ; acc = txdata
         sacl   *               ; txdata_main/aux = transmit data

****************************************************************************
**
*       drx_clear_orig_mb1 and drx_clear_answ_mb1 handle receive HDLC functions
*       for V.120 and PPP for mailbox 1.
*       This is a possible state of rxvect/2
*
*       inputs  dp = 7
*       ar(arp) = ar1
*               ar1 = *(8 bit rceive sample from the link)
*
*       outputsdp = 7
*               ar(arp) = ar1
****************************************************************************
**

drx_clear_orig_mb1
drx_clear_answ_mb1              ; dp = 7
        ldp     #6              ; dp = 6
        lacl    *               ; acc = receive data

        .if     PCM code
        mar     *,ar2
        lar     ar2,#arcflg
        bit     *,farc,ar1      ; stos?
        cc      bit_reverse8,tc ; Prepare "digital" sample for transmission
        .endif


        and     rxmask          ; mask to 8 or 7 bits
      sacl      rxdata          ; and save for transport to SV
```

```
        .if     PCM code
        mar     *,ar2
        lar     ar2,#isdnflg
        bit     *,fstosscr,ar1  ; is it an PCM code server to server call w/scr?
        cc      descram_v32answ,tc ; call PCM code server to server RX routine

*       bit     isdnflg,fstoszbs ; PCM code stos call w/zero byte supression
*       cc      rx_zbs,tc       ; call rx zero byte supression routine
        .endif

        call    queue_rxd,*,ar1 ; send receive data to supv. (input dp = 6)
        ldp     #7              ; output dp = 7
        ret

*****************************************************************************
**
*       dtx_clear_orig_mb2 and dtx_clear_answ_mb2 handle xmit HDLC functions
*       for V.120 and PPP for mailbox 2
*       This is a possible state of txvect/2
*
*       inputs  dp = 7
*       ar(arp) = ar1
*               ar1 = *(8 bit sample to transmit to the link)
*
*       outputsdp = 7
*               ar(arp) = ar1
*****************************************************************************
**

dtx_clear_orig_mb2
dtx_clear_answ_mb2                      ; dp = 7

        ldp     #isdn_dp2       ; use MB2
        sar     ar1,tempx       ; tempx = ptr to xmit sample

        call    get_txd,*,ar1   ; get data from supv. (must not use tempx)
                                ; dp = isdn_dp2
        .if     PCM code
        lacl    txdata
        and     txmask          ; mask off extra bits (for scrambler)
      sacl    txdata          ; save transmit data

        bit     isdnflg,fstosscr ; PCM code server to server call with scrambler?
        cc      scram_v32orig,tc ; call PCM code server to server TX routine
        .endif

        lar     ar1,tempx       ; ar1 = ptr to xmit sample

        lacl    txdata          ; acc = txdata
```

```
        retd
         ldp    #7              ; return dp = 7
         sacl   *               ; txdata_main/aux = transmit data

****************************************************************************
**
*       drx_clear_orig_mb2 and drx_clear_answ_mb2 handle receive HDLC functions
*       for V.120 and PPP.
*       This is a possible state of rxvect/2
*
*       inputs  dp = 7
*       ar(arp) = ar1
*               ar1 = *(8 bit rceive sample from the link)
*
*       outputsdp = 7
*               ar(arp) = ar1
****************************************************************************
**

drx_clear_orig_mb2
drx_clear_answ_mb2                  ;


        ldp     #isdn_dp2       ; yes, use MB2
        lacl    *               ; acc = receive data
        and     rxmask          ; mask to 8 or 7 bits
      sacl      rxdata          ; and save for transport to SV

        .if     PCM code
        mar     *,ar2
        lar     ar2,#isdnflg
        bit     *,fstosscr,ar1  ; is it an PCM code server to server call w/scr?
        cc      descram_v32answ,tc ; call PCM code server to server RX routine
        .endif

*       bldd    #rx_crc13,tempz         ; move rx_crc from isdn_dp2 to tempz
*       call    queue_rxd_mb2,*,ar1 ;send receive data to SV(inpt dp=isdn_dp2)
        call    queue_rxd,*,ar1 ;send receive data to SV(inpt dp=isdn_dp2)
        ldp     #7              ; output dp = 7
        ret

                .end
```

We claim:

1. A method for determining characteristics of a data communication channel between first and second data communication devices comprising, in combination:

sending a relatively low power digital probe signal over said channel from said first to said second data communication devices;

sending a second digital probe signal corresponding to an analog signal having a relatively high-frequency signal with a time-varying dc component from said first to said second data communication devices; and

detecting a received signal at said second data communication device, said received signal corresponding to said probe signals sent by said first device, and determining whether said received signal varies from a predetermined standard.

2. A method as claimed in Claim 1, wherein said relatively high frequency signal is greater than 3,300 Hertz.

3. A method as claimed in Claim 1, wherein said relatively high frequency signal is substantially equal to 4,000 Hertz.

4. A method as claimed in claim 1, wherein said relatively high frequency signal has an amplitude of approximately one-half of a maximum amplitude permitted for said channel.

5. A method as claimed in claim 1, wherein said probe signal is synchronized between said first and said second data communication device.

6. A method as claimed in claim 5, wherein said probe signal includes a total number of bytes substantially equally divisible by 6.

7. A method as claimed in claim 1 wherein said method further comprises sending a third, substantially all-zero, probe signal.

8. A method as claimed in claim 1 wherein said step of determining whether said received signal varies from a predetermined standard includes determining whether robbed bit signaling has affected a probe signal.

9. A method as claimed in claim 8, wherein said step of determining whether said received signal varies from a predetermined standard further comprises at least one of: determining a number of robbed bits, and determining a location of a robbed bit.

10. A method as claimed in claim 1 wherein said step of determining whether said received signal varies from a predetermined standard includes determining whether digital ones have been inserted in a probe signal.

11. A method as claimed in claim 1 wherein said step of determining whether said received signal varies from a predetermined standard includes determining whether a probe signal has passed through a digital pad.

12. A method as claimed in claim 1 wherein said step of determining whether said received signal varies from a predetermined standard includes monitoring said received signal for a

digital signal corresponding to an analog signal having a relatively high-frequency signal with a time-varying dc component.

13. A method as claimed in claim 1, wherein said second probe signal comprises a sequence of digital codewords.

14. A method as claimed in claim 13, wherein said sequence of digital codewords comprises 256 distinct codewords.

15. A method as claimed in claim 14, wherein said 256 distinct codewords correspond to a series of levels associated with a 256 level quantizer.

16. A method as claimed in claim 1, wherein said relatively low power digital probe signal is of sufficient length so that an average power of at least said low power digital probe signal and said second digital probe signal is less than a predetermined threshold.

FIG. 1

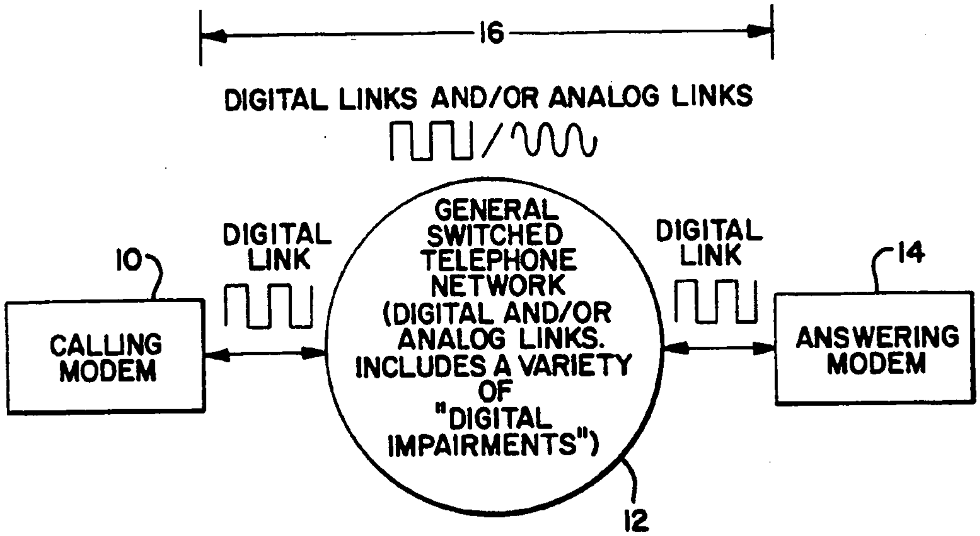

16
DIGITAL LINKS AND/OR ANALOG LINKS
10
DIGITAL LINK
CALLING MODEM
GENERAL SWITCHED TELEPHONE NETWORK (DIGITAL AND/OR ANALOG LINKS. INCLUDES A VARIETY OF "DIGITAL IMPAIRMENTS")
DIGITAL LINK
14
ANSWERING MODEM
12

FIG. 2

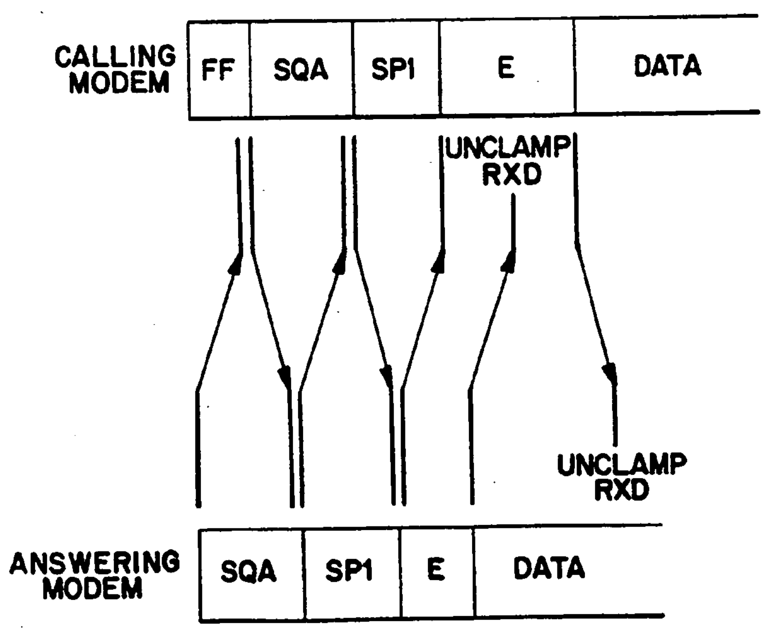
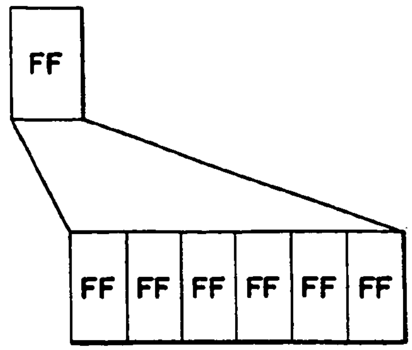
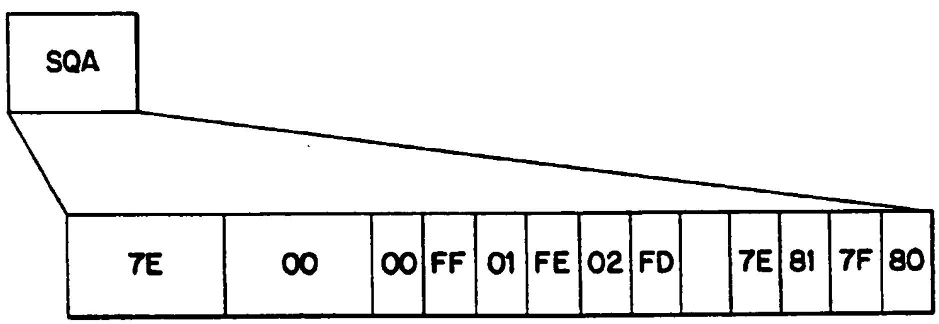

CALLING MODEM
FF
SQA
SP1
E
DATA
UNCLAMP RXD
UNCLAMP RXD
ANSWERING MODEM
SQA
SP1
E
DATA

FIG. 3



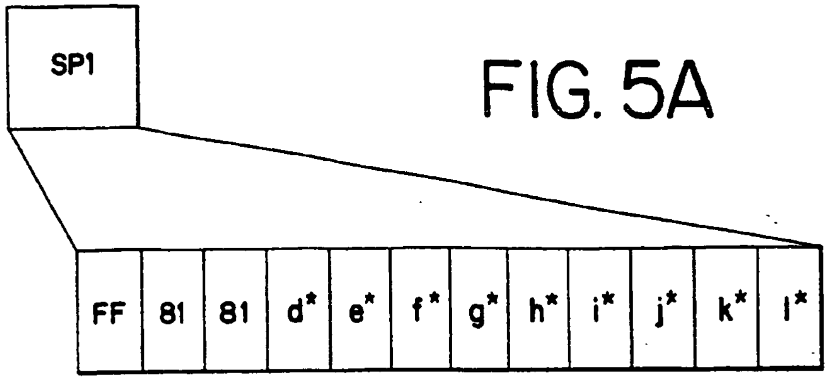
THE FF PATTERN IS DEFINED AS TRANSMITTING THE HEX BYTE FF REPEATEDLY

FIG. 4



THE SQA PATTERN BEGINS WITH 7+6n BYTES OF 7E, WHERE n= 290. THIS IS FOLLOWED BY 7 BYTES OF 00. THIS IS FOLLOWED BY BYTES GENERATED BY AN INCREASING COUNTER FROM 00 TO 7F WHICH ARE INTERLEAVED WITH BYTES GENERATED BY A DECREASING COUNTER FROM FF TO 80. THE FULL SEQUENCE, AS DESCRIBED ABOVE IS ONE FULL PERIOD OF SQA. THIS SEQUENCE WILL REPEAT AS LONG AS THE TRANSMITTER IS IN THE SQA STATE.

FIG. 5A



THE SP1 SEQUENCE CONSISTS OF ONE BYTE OF FF, TWO BYTES OF 81, AND 9 BYTES (d*-l*) OF INFORMATION. THE INFORMATION BYTES ALL HAVE THE MOST SIGNIFICANT BIT SET TO 0 AND THE LEAST SIGNIFICANT BIT SET TO 1, AND THE MIDDLE SIX BITS CONTAIN INFORMATION (DEFINED IN NEXT FIGURE.)

FIG. 5B

d*: VERSION INFORMATION: 000000
e*: RESERVED FOR FUTURE USE: 000000
f*: ROBBED BIT SIGNALING POSITION: 1=RBS ON THIS BIT, 0=NO RBS ON THIS BIT
g*: MISCELANEOUS INFO: (BIT b0 IS THE LEAST SIGNIFICANT BIT)
  b6 = 0 (RESERVED FOR FUTURE USE)
  b5 = 0 (RESERVED FOR FUTURE USE)
  b4 = 1 TO USE ZERO BYTE SUPRESSION, 0 TO NOT USE zbs
  b3 = 1 TO SCRAMBLE THE DATA, 0 NOT TO SCRAMBLE THE DATA
  b2 = 1 IF THE CHANNEL IS "RESTRICTED", 0 IF THE CHANNEL IS UNRESTRICTED.
  b1 = 1 IF 64kbps SPEED IS SELECTED, 0 IF 64kbps SPEED NOT SELECTED
h*: SPEED BITS:
  b6 = 1 IF 62.6Kbps SELECTED. 0 = NOT SELECTED
  b5 = 1 IF 61.3Kbps SELECTED. 0 = NOT SELECTED
  b4 = 1 IF 60.0Kbps SELECTED. 0 = NOT SELECTED
  b3 = 1 IF 58.6Kbps SELECTED. 0 = NOT SELECTED
  b2 = 1 IF 57.3Kbps SELECTED. 0 = NOT SELECTED
  b1 = 1 IF 56.0Kbps SELECTED. 0 = NOT SELECTED
i*: CRC 4 BITS: 000CRCa1 (FIRST 4 BITS)
j*: CRC 4 BITS: 000CRCb1 (SECOND 4 BITS)
k*: CRC 4 BITS: 000CRCc1 (THIRD 4 BITS)
l*: CRC 4 BITS: 000CRCd1 (FOURTH 4 BITS)

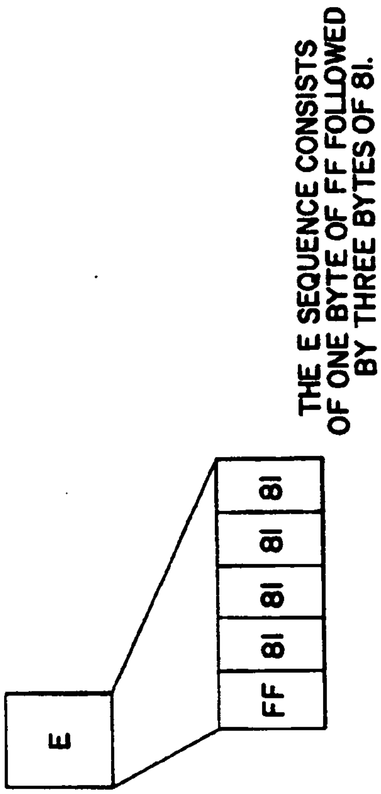

E
FF
8I
8I
8I
8I
THE E SEQUENCE CONSISTS OF ONE BYTE OF FF FOLLOWED BY THREE BYTES OF 8I.

FIG.6

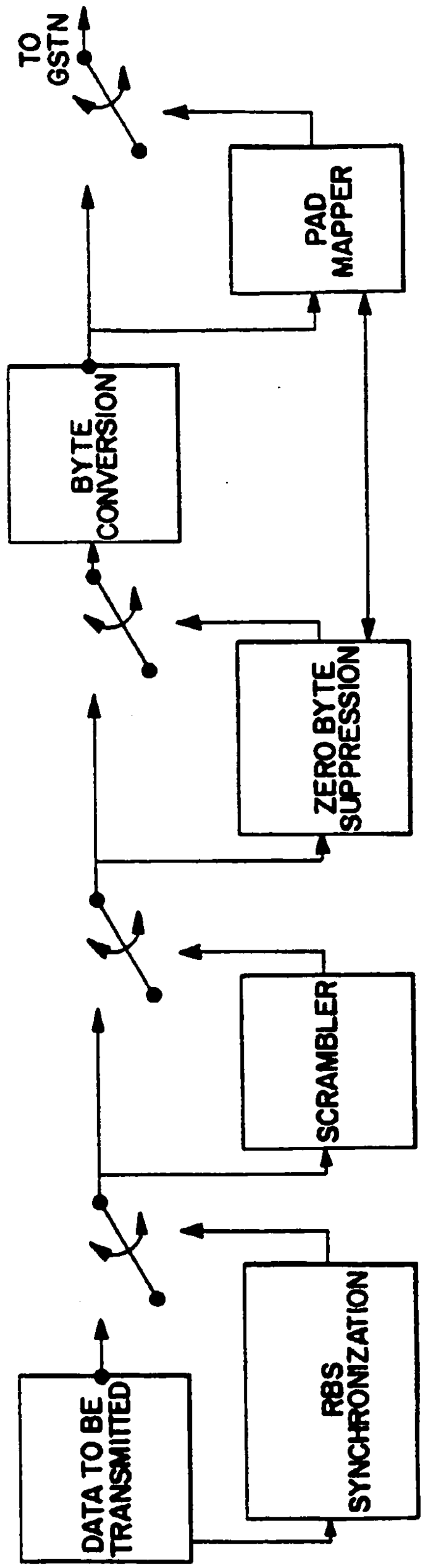

DATA TO BE TRANSMITTED
RBS SYNCHRONIZATION
SCRAMBLER
ZERO BYTE SUPPRESSION
BYTE CONVERSION
PAD MAPPER
TO GSTN

FIG. 7

INTERNATIONAL SEARCH REPORT

Inte onal Application No
PCT/US 98/04271

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04J3/14 H04J3/12 H04Q11/04

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H04J H04Q H04M H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 398 234 A (O'CONNELL ANNE ET AL) 14 March 1995 see abstract see column 1, line 19 - column 2, line 2 see column 3, line 10 - line 28 --- -/--	1,8-10

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
18 June 1998	25/06/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Koukourlis, S

1

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 98/04271

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	INGLE J F: "HOW TO DETECT FRAME SLIPS IN VOICEBAND PCM CHANNELS" DATA COMMUNICATIONS, vol. 17, no. 11, October 1988, NEW YORK, US, page 205/206, 209/210, 217/218, 221/222 XP000032541 see page 205, left-hand column, paragraph 3 see page 209, left-hand column, paragraph 3 - right-hand column, paragraph 1 see page 218, right-hand column, paragraph 5 - page 221, right-hand column, paragraph 4	1-3,5,6, 8,9,12
A	GB 1 417 627 A (HASLER AG) 10 December 1975 see page 1, right-hand column, line 50 - line 68 see page 2, left-hand column, line 36 - right-hand column, line 111 see claim 1	1,12-15
A	US 4 730 302 A (FUERLINGER FRANZ ET AL) 8 March 1988 see abstract see column 4, line 55 - line 68	1,10
A	US 4 161 627 A (AMANN BERTRAM) 17 July 1979 see abstract see column 1, line 35 - column 2, line 34	1-3,7, 12-15
A	DE 43 43 982 C (SIEMENS AG) 17 August 1995 see abstract	1

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inte ional Application No
PCT/US 98/04271

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 5398234	A	14-03-1995	NONE		
GB 1417627	A	10-12-1975	CH	551112 A	28-06-1974
			DE	2346607 A	02-05-1974
			FR	2204086 A	17-05-1974
			JP	49075253 A	19-07-1974
			NL	7314164 A	26-04-1974
			SE	7314401 A	25-04-1974
US 4730302	A	08-03-1988	NONE		
US 4161627	A	17-07-1979	DE	2659512 B	24-05-1978
			FR	2376563 A	28-07-1978
			GB	1556292 A	21-11-1979
DE 4343982	C	17-08-1995	NONE		